`remarks` — Machinery for Marking Student Work

Oleks Shturmov[1,2,3][0000−0001−5963−8509] and
Michael Kirkedal Thomsen[1,2,4][0000−0003−0922−3609]

[1] Department of Computer Science, University of Copenhagen, Denmark
[2] Department of Informatics, University of Oslo, Norway
[3] oleks@oleks.info
[4] michakt@ifi.uio.no

Abstract. `remarks` is an open-source suite of tools for marking student work. It has been in moderate use for assessing semi-structured submissions in a handful of Computer Science courses with hundreds of students since 2016. The contemporary approach to systematically collaborate on such assessments is to use general-purpose administrative software (e.g., spreadsheets, text files or documents, Emacs Org-mode). Since this software has not been expressly designed for the purpose, it tends to require non-trivial technical skill to achieve but mediocre technical support for the assessment process. In particular, it is hard to achieve (1) support for criteria-based analytic marking, without precluding (2) holistic, free-form justifications for the given marks, while enabling (3) decentralized collaboration with a reliable and transparent synchronization mechanism. `remarks` has been expressly designed to meet these criteria. Guided by the theory of assessment and evaluation in higher education, it has grown from a series of scripts surrounding general-purpose administrative software to a dedicated tool suite for marking student work. With `remarks`, assessing a unit of work constitutes filling in a document whose structure is guided by, but not limited to a chosen marking scheme. Assessors can use a contemporary source code revision control system (e.g., Git) to collaborate on these documents; making conflict resolution explicit and unsurprising. Using one document per unit of work, further reduces conflicting edits, in general. The `remarks` tool can then derive summative and descriptive assessments from such documents. We describe and justify the design of `remarks`, report on our experiences with the approach, and invite you to help develop the ideas further.

Keywords: Criteria-based assessment · Analytic scoring · language-oriented programming.

1 Introduction

Computer Science courses tend to assess student learning by assigning students practical work (e.g., take-home assignments or exams) and asking them to document it in semi-structured writing. Students submit computer programs that might have some expected structure or behavior, and possibly a technical report, having perhaps an expected layout, documenting their work further.

Educators then engage in assessing the deliveries in relation to desired learning outcomes. They often can get help from teaching assistants to ensure a speedy assessment, and often must convene with external examiners to ensure validity and reliability of any final assessment. Overall, for a fair-sized Computer Science course, having hundreds of students, a dozen or so human assessors may be involved in the process.

Coordinating their effort is no easy feat. Firstly, it is best if there is a unified marking scheme, such that each submission is treated similarly even if marked by different assessors. Secondly, the marks they give should be justified. However, having an overly fine-grained marking scheme can both make it hard to apply to every submission, and substantially increase the workload for the assessor. Leaving out marking schemes allows assessors to justify their decisions as they see fit, but at the cost of consistency across the assessment team. Alas, we also run the risk that no justification may be deemed fit by some assessors. Thirdly, making an assessment requires a non-trivial amount of mental effort, which can be hard for humans to exert for long stretches of time. It seems best to let assessors work at their own pace, subject only to a common deadline.

To these ends, we present `remarks`—a suite of tools for marking student work. `remarks` is an open-source project[5], initiated at the Department of Computer Science at the University of Copenhagen (DIKU) in 2016. `remarks` has seen moderate use for marking free-form written assignments and programming tasks in both coursework and exams in courses such as Computer Systems, IT Security, Software Development, and Advanced Programming. The feedback overall is positive, in that `remarks` is simple for the assessors to use, and tends to lead to assessments with a seemingly high level of validity and reliability.

At present, assessors work with `remarks` by (1) filling out plain-text files reminiscent of Markdown, but substantially richer in structure, (2) synchronizing those files via git, and (3) using `remarks` tool suite to arrive at collective assessments and analyses. We are working a web-based interface for `remarks`, which would allow for even more prompt collaboration between the assessors.

`remarks` was inspired by Markdown and Emacs Org-mode, and has grown from a collection of shell scripts to a full-blown domain-specific language. As such, the design of `remarks` follows a language-oriented approach [8, 13], where the goal is to eloquently capture the domain of marking student work.

The rest of this paper is structured as follows: Section 2 introduces the linguistic concepts used in both the design of `remarks`, and the rest of this paper. Sections 3 and 4 present the resulting domain model and concrete syntax of `remarks` files. Though technical, they illustrate how the domain comes across in the design and implementation of `remarks`. If you read these, we also refer you to Appendix D for a quick overview of the file system and command-line interface of `remarks`. Section 5 goes over our experience with the tool, together with a recent (small) user study. Section 6 concludes and discusses future work.

[5] `https://github.com/DIKU-EDU/remarks`

2 Background

Assessing student learning is a fundamental activity in higher education, consisting of sampling student work and provisioning appraisals for it in relation to intended learning outcomes (ILOs) [2, 11]. It is through assessment that educators gauge, steer, and evaluate student learning. Students on the other hand, get feedback about their learning, and receive grades which go towards their learning certificates, having significant career implications. As such, assessment plays a key role in both learning activities and evaluation of learning outcomes; assessment should happen with a high degree of validity and reliability.

An invalid assessment measures something other than in-how-far the ILOs have been attained. An unreliable assessment cannot be replicated in diverging circumstances (e.g., when done by another assessor, or via a different mode of assessment). Validity and reliability are especially important in the context of grading, but it seems only healthy to also strive for it while teaching is ongoing, to keep teaching and learning duly on track. Since assessment is about measuring human performance, the process is generally riddled with many potential threats to validity (and reliability) [2, 12].

Computer-assisted assessment (CAA) has long been promised to enable assessments (e.g., of student learning) with a high degree of reliability and validity in a scalable and efficient manner [2–4]. Unfortunately, these promises have thus far seemingly failed to materialize [4]. This is anecdotally evidenced by the fact that while most educators today do use computers, they do not tend to use any specialized software, but instead opt for general-purpose administrative software. We discuss one commonly used tools in Section 2.1 and Appendix E.

CAA can come with various levels of automation. In this work, we will focus on the use of computers as aids in primarily human-driven assessment. We will refer to the humans in question as "assessors"[6]. Some assessors will be educators, and the rest either teaching assistants or external examiners. All assessors have expertise in the domain being taught, but educators also have formal training in teaching. Educators bear the responsibility for the success of the learning process and specify ILOs. The other assessors aid to make valid and reliable assessments.

Technical fields, such as Computer Science, often require that students attain certain knowledge of theoretical concepts, as well as certain analytical competences and technical skills throughout a course. Assessment of such ILOs often happens by asking students to conduct practical work, and document it in semi-structured writing (e.g., as computer programs and/or technical reports).

There are several key issues to consider to ensure validity with this mode of assessment. Firstly, students must not have to spend time on unrelated ground work in order to do the assignment—it should be properly aligned with course prerequisites and the ILOs. Secondly, the assignment should be sufficiently unique so that it may serve as an exhibit of the student's analytical abilities, rather than rote learning. Thirdly, there is bound to be a gap between the formulation of the ILOs and the (unique) assignment. We can bridge this gap by devising dedicated

[6] Alternative, synonymous names are "appraisors" or "markers" [10].

evaluative criteria, which pinpoint the qualities that a submission must have in order to serve as an exhibit of the attainment of the ILOs. For instance, a desired learning outcome may be "knowledge of synchronization primitives in a low-level systems programming language", while an evaluative criteria could be "uses the provided lock when accessing the shared struct".

Devising evaluative criteria alongside the assignment helps both ensure constructive alignment with the ILOs, and arrive at a marking scheme for submissions. A marking scheme suggests the criteria to evaluate for every submission, how to judge each criterion, and the weight it plays in the overall assessment. Having a marking scheme aids in ensuring the validity and reliability of an assessment, since each submission is treated with similar scrutiny.

Marking schemes are sometimes also called "scoring guides" or "rubrics" [5, 9, 10]. Scoring guides in particular suggest that for each evaluative criteria, the assessor must assign a score. These scores are are then combined, analytically to form an overall score for the submission. Rubrics, on the other hand, tend to focus more on illustrating what constitutes a good, mediocre, or bad exhibit along the evaluative criteria. In our view, both qualities are important, and we will not delve on these distinctions further. Instead, we focus on having marking schemes, reminiscent of scoring guides and rubrics at the same time.

Finally, it is commonplace to distinguish between formative and summative assessment. In formative assessment, the appraisal is delivered as contextualized prose, directly to the student. The purpose is to "form" better student understanding of the subject matter, as well as "form" subsequent learning activities (e.g., a student is asked to resubmit). Summative assessment on the other hand is delivered as a grade, placing the student on a grading scale. Students tend to care more about their individual grades, while the educator may be interested in how the class is doing as a whole.

Summative assessments are readily comparable. Students with the same grade exhibit a comparable level of attainment of the ILOs from a holistic perspective. Summative assessment however does not provide insight into the student's attainment of the individual learning outcomes. Students with the same grade may understand different parts to different extent. Although we can glean the difference in their understandings from formative assessments, if any, these are not readily composable.

Lesser known is diagnostic assessment, where the appraisal consists of a collection assessments, providing insight as to the attainment of some individual learning outcomes. For instance, a filled out marking scheme constitutes a diagnostic assessment. A diagnostic assessment is perhaps primarily of interest to the educator in order to better align subsequent teaching, but it can also be a good form of feedback for the student, as a more structured alternative to purely formative or purely summative feedback.

2.1 Example: Collaborative Online Spreadsheets

When faced with an assessment task, like the ones outlined above, many educators will bring out a collaborative online spreadsheet.

	A	B	C	D	E	F	G
1		**Theory(50)**			**Practice(50)**		**Total(100)**
2		**T1(10)**	**T2(20)**	**T3(20)**	**P1(25)**	**P2(25)**	
3	**S1**						`=SUM(B3:F3)`
4	**S2**						`=SUM(B4:F4)`
	⋮						

Table 1. Sample marking scheme spreadsheet for submissions S1, S2, ...

Table 1 shows an example of such a spreadsheet. We dedicate a row for each submission, and a column for each (sub)task of the assignment. The assessors then give a mark to each cell in the resulting matrix. Once done, the columns are analytically combined to form an overall score for each submission.

This (simplified) example suffers from a number of drawbacks. Some are methodological (**SM**), others technical (**ST**), and others still have to do with collaboration (**SC**). Methodological drawbacks can usually be compensated for by writing up additional, but technically disjoint documents. Technical drawbacks can often be remedied by leveraging additional technical features of common spreadsheet software. Collaboration drawbacks stem from the fact that spreadsheets are usually not subject to rich revision control systems [7, 14].

SM1 It is not clear what qualities in a submission should yield what number of points for each column in the table.
SM2 It is not clear where and how an assessor should leave comments to justify the marks that they give.
SM3 It is not clear how the marks in the table relate to the ILOs.
ST1 Maximum points are not enforced, here they are mere text.
ST2 It is common to apply additional weights to the rows and/or columns. For instance, the workload of filling in the marks could be divided among many assessors, such that each assessor either marks a subset of the columns for all submissions, or all columns for a subset of the submissions.
ST3 It is common to allow bonus or foul points, to account for work done, or omissions, beyond what the marking scheme covers.
SC1 Users do not structure their changes in terms of well-formed commits; changes are tracked automatically, leaving ample room for accidental edits.
SC2 Spreadsheets are not easily compared and merged; automatic merging is not solvable in general, unless the edit operations are suitably constrained [14].

3 Domain Model

Given the terminology defined in Section 2, and motivated by the examples therein, we now present the domain model of `remarks`. We present it as an abstract syntax tree. This can be made concrete in a domain-specific language, as we discuss in Section 4, or a data model in a more conventional DBMS, or

something third entirely. These latter options are left as future work. For an in-depth discussion of language-oriented programming, and the notation used in this section and the next, we refer the interested reader to Appendix C.

Assessing a unit of work with `remarks` constitutes making a judgement. Judgements can be hierarchically structured, where sub-judgments serve to analytically and descriptively justify their super-judgements.

A judgement consists of a header and some justification[7]. Judgements can be nested, and so they have an associated depth (here written as n).

$$\langle \mathit{Judgement} \rangle_n ::= \langle \mathit{Header} \rangle_n \ \langle \mathit{Justification} \rangle_n$$

The header has some identifying information, to distinguish the judgement from other judgements. In its basic form, a judgement is summative—an assessor can assign a number of points out of a set maximum number of points. This too occurs in the header. However, the assessor does not need to set the points, if they can be analytically deduced from sub-judgements (e.g., as a weighted sum). Finally, as a special case, it might not make sense to perform any judgement (e.g., if the student did not do the unit of work).

$$\begin{aligned} \langle \mathit{Header} \rangle_n &::= \langle \mathit{ID} \rangle \ \langle \mathit{MaybePoints} \rangle \ \langle \mathit{MaxPoints} \rangle \\ \langle \mathit{MaybePoints} \rangle &::= \text{`}\texttt{Analytic}\text{'} \mid \langle \mathit{GivenPoints} \rangle \mid \text{`}\texttt{Invalid}\text{'} \end{aligned}$$

To holistically, descriptively, and analytically justify a summative assessment, the assessor can provide a sequence of remarks and sub-judgements:

$$\langle \mathit{Justification} \rangle_n ::= \langle \mathit{Remark} \rangle_n^* \ \langle \mathit{Judgement} \rangle_{(n+1)}^*$$

Sub-judgements are regular judgements, but with an increased depth. A remark is a textual comment with a mood indicator. Remarks too can be hierarchically structured for more detailed holistic justifications.

$$\begin{aligned} \langle \mathit{Remark} \rangle_n &::= \langle \mathit{Mood} \rangle \ \langle \mathit{Text} \rangle \ \langle \mathit{Remark} \rangle_{(n+1)}^* \\ \langle \mathit{Mood} \rangle &::= \text{`}\texttt{Positive}\text{'} \mid \text{`}\texttt{Negative}\text{'} \mid \text{`}\texttt{Mixed}\text{'} \\ &\quad \mid \text{`}\texttt{Stuctural}\text{'} \\ &\quad \mid \text{`}\texttt{Warning}\text{'} \mid \text{`}\texttt{Impartial}\text{'} \end{aligned}$$

The "mood" '`Structural`' is not a mood, but rather indicates a bullet-point which may have sub-remarks. The moods '`Warning`' and '`Impartial`' are useful in multi-pass assessments (e.g., where assessments are subsequently validated by educators and external examiners). The first serves to warn the subsequent assessor, the second suggests that there is something worthy of note, but it is left for the successor to figure out the consequences.

The assessor can choose to have just remarks, in which case the judgement is (modulo mood indicators) holistically justified, or just have sub-judgements,

[7] We admit additional properties on judgements, but we omit this for brevity.

in which case it is analytically justified; or choose to have both. The latter case is particularly useful to handle cases where making preset sub-judgements only would not fully justify the summative assessment (e.g., bonus points).

Both $\langle \mathit{Judgement} \rangle$ and $\langle \mathit{Remark} \rangle$ roughly correspond to an evaluative criterion. The difference lies merely in the form of assessment.

4 Plain-Text Format

The `remarks` plain-text format is inspired by Markdown[8]. As such, the $\langle \mathit{Header} \rangle_n$ of a $\langle \mathit{Judgement} \rangle_n$ begins with n '`#`' symbols, reminiscent of a Markdown heading, and ends with a special suffix—maybe some points out of a given maximum number of points. Line breaks are not allowed in a judgement header.

$$\langle\!\langle \langle \mathit{Header} \rangle_n \rangle\!\rangle ::= \text{`\#'}^n \ \langle\!\langle \langle \mathit{ID} \rangle \rangle\!\rangle \ \text{`:'} \ \langle\!\langle \langle \mathit{MaybePoints} \rangle \rangle\!\rangle \ \text{`/'} \ \langle\!\langle \langle \mathit{MaxPoints} \rangle \rangle\!\rangle$$

$\langle\!\langle \langle \mathit{ID} \rangle \rangle\!\rangle$ is a non-empty sequence of characters, except line break and '`:`'. The depth n and the text $\langle\!\langle \langle \mathit{ID} \rangle \rangle\!\rangle$ uniquely identify a judgement.

$\langle\!\langle \langle \mathit{MaxPoints} \rangle \rangle\!\rangle$ is a non-empty sequence of digits, optionally followed by a decimal point and exactly two digits after the decimal point. `remarks` does not allow arbitrary floating-point values by design—we do not want to deal with round-off errors, and other potential issues relating to use of floating-values in assessments. Internally, points are represented as integers.

$\langle\!\langle \langle \mathit{MaybePoints} \rangle \rangle\!\rangle$ is either a sequence as $\langle\!\langle \langle \mathit{MaxPoints} \rangle \rangle\!\rangle$ above, indicating $\langle \mathit{GivenPoints} \rangle$, the empty string ϵ indicating '`Analytical`', or the character '`-`', indicating '`Invalid`'.

$$\langle\!\langle \langle \mathit{MaybePoints} \rangle \rangle\!\rangle ::= \epsilon \mid \langle\!\langle \langle \mathit{GivenPoints} \rangle \rangle\!\rangle \mid \text{`-'}$$

As with Markdown, a $\langle\!\langle \langle \mathit{Header} \rangle_n \rangle\!\rangle$ is followed by a line-break, when seen in the context of a judgement. Properties too are separated by line breaks.

$$\langle\!\langle \langle \mathit{Judgement} \rangle_n \rangle\!\rangle ::= \langle\!\langle \langle \mathit{Header} \rangle_n \rangle\!\rangle \ \text{`\textbackslash n'} \ \langle \mathit{Justification} \rangle_n$$

The following `remarks` file corresponds to a submission row in Table 1:

```
# Theory: /50
## T1: /10
## T2: /20
## T3: /20
# Practice: /50
## P1: /25
## P2: /25
```

[8] In fact, `remarks` was initially a collection of shell scripts that manipulated Markdown-styled files.

Remarks are again inspired by Markdown bullet-point lists. However, instead of meer bullets, we can use mood indicators to indicate whether the item is positive, negative, etc. Individual remarks are separated by line breaks.

$$\langle\!\langle \langle \mathit{Remark} \rangle_n \rangle\!\rangle ::= \text{`\,'}^n \ \langle\!\langle \langle \mathit{Mood} \rangle\!\rangle \ \text{`\,'} \ \langle\!\langle \langle \mathit{Text} \rangle\!\rangle \ \left\langle\!\!\left\langle \langle \mathit{Remark} \rangle_{(n+1)} \right\rangle\!\!\right\rangle^*$$

The mood marks are defined as follows:

$$\begin{aligned}
\langle\!\langle \text{`}\texttt{Positive}\text{'} \rangle\!\rangle &::= \text{`}\texttt{+}\text{'} & \langle\!\langle \text{`}\texttt{Structural}\text{'} \rangle\!\rangle &::= \text{`}\texttt{*}\text{'} \\
\langle\!\langle \text{`}\texttt{Negative}\text{'} \rangle\!\rangle &::= \text{`}\texttt{-}\text{'} & \langle\!\langle \text{`}\texttt{Warning}\text{'} \rangle\!\rangle &::= \text{`}\texttt{!}\text{'} \\
\langle\!\langle \text{`}\texttt{Mixed}\text{'} \rangle\!\rangle &::= \text{`}\texttt{\~{}}\text{'} & \langle\!\langle \text{`}\texttt{Impartial}\text{'} \rangle\!\rangle &::= \text{`}\texttt{?}\text{'}
\end{aligned}$$

The plain-text format is further supported by the file system and command-line interface of `remarks`. See Appendix D for a further overview.

5 Experience

`remarks` has in different aspects been actively used at the Department of Computer Science, University of Copenhagen (DIKU) since 2016. It has mainly been used on the 15 ECTS second-year bachelor course Computer Systems[9] (CompSys) with about 180 students, 7 teaching assistants (TAs) and 3 teachers and 4 external exam examiners. CompSys was created as a course in 2016. `remarks` was also used from 2016 to 2022 on the elective 7.5 ECTS third-year bachelor course IT-security[10] (ITS) with about 100 students, 3 TAs and 2 teachers.

The usage can categorised as following:

- Marking in-course assignment submissions.
- Marking exam assignment submissions.

5.1 Scenarios

We will in the following detail the two usage scenarios from DIKU. We will note that this is not a full list and `remarks` can easily be used in other situations. The authors have themselves used `remarks` for simple exam corrections and assessments of exam complaints. But these two scenarios shows the full benefit when more people need to collaborate.

[9] Links to Computer Systems:
- Course description: `https://kurser.ku.dk/course/ndab16005u/`.
- Latest course page: `https://github.com/diku-compSys/compSys-e2024`

[10] Links to IT-Security:
- Course description: `https://kurser.ku.dk/course/ndaa09025u`.
- Latest course page: `https://github.com/diku-its/e2024`

Marking in-course assignment submissions

Situation: Both CompSys and ITS had multiple TAs that individually mark a number of submissions for each assignment. The TAs give written feedback to the students, score and pass/fail the submission.

Problem: The assignments (especially on CompSys) were quite large with students working over 2-3 weeks, but still had underlying learning goals related to the course schedule. With 4 to 7 TAs to assess the submissions and give feedback to the students, this requires a significant amount of coordination either during marking, or before assessment is given to students. However, the TAs are also full-time students that mark at different paces, so with coordination amongst the TAs, the assessment for all students can be delayed. Having little to no coordination can result in different levels amongst the TAs.

On both CompSys and ITS we observed a reduction in the number of student complaints about inconsistent, or low quality TA marking, as we have gradually adopted and improved our use of `remarks` over the years.

Approach: On both courses the teacher writes the `remarks` files for the assignments and the TAs assess and give feedback to the students based on them. The files contain detailed evaluative criteria, derived from overall ILOs for the course, specialized for the assignment. This provides the TAs with a high-level overview of the assignment. The assessments in the `remarks` files made by the TAs are also used for potential resubmissions, by updating the assessments. The `remarks` files are not shared with the students, but the most significant parts have overlapping text in the assignment texts. An example of a `remarks` file used in CompSys is included in Appendix A.

We try to create the `remarks` files with the assignment. As for the assignment text, the `remarks` files are discussed with the TAs prior to start of the assignment; the intention is that TAs can assist students to focus their assignment work.

When assessing, the TAs annotate the `remarks` file and give a percentage that then is translated to a to a number of points (0-4) or pass/fail of a predefined scale. The TAs also use the `remarks` entries to guide the written feedback that they give to the students.

Marking exam assignment submissions

Situation: The final grade of CompSys is based on an evaluation by both an internal evaluator (often course teacher) and an external examiner. Due to the number of students there are 4 external examiners. The size of the course[11] and number of students also means that internal evaluation is separated between several teachers.

[11] The topics of CompSys includes computer architecture, operating systems, computer networks, and simple IT-security.

Problem: Assessing the exams of a course with many students is a big task with many different persons involved. Today, to free up time for the teachers for student confrontations, TAs give a first assessment, followed by the teachers and external examiners. On CompSys this results in a total of 14 to 16 persons involved in the exam assessment. This requires much coordination to avoid variation in evaluations and using less time on discussing grading of individual students between the internal evaluators and the external examiners.

Approach: It is the teachers (relating to the different topics on the course) that make the `remarks` files for the exam evaluation. We have tried to define the `remarks` files such that they fit the ILOs of the course. We can then use these when creating the exam, to evaluate how well we test these. In later years as the exam format has stabilised (though the individual questions are still updated), the `remarks` files have also become quite stable. After the TAs have tested the exam set, we also ask them how well the `remarks` files fit, to get an assessment of how well our ILOs are tested by the exam questions. Finally, both exam sets and `remarks` files are sent to the external examiners for information and comments.

After students submit their exam assignments, the TAs make an initial assessment of the answers. This is strictly based on the points in the `remarks` files and each TA will assess one or two questions for all students. The intent of this is to get a consistent assessment of each question without considering the assignment as a whole. Afterwards the teachers (internal evaluators) will look though the assessment and using `remarks` adjust the level if needed. The external examiners are assigned distinct subsets of the students. The examiners gets the full assessment from `remarks`; there are several formats for this, but with `remarks` we can insert the individual assessments as comments in the pdf-files that are handed in or give a detailed interactive table with the assessments.

As a result, TAs and internal evaluators give a "vertical" (per question) assessment, while the external examiners give a "horizontal" (per student) assessment (see also Table 1). This approach requires much coordination, which is made easy with `remarks`.

5.2 Course Evaluations

All courses at the University of Copenhagen are subject to evaluation by their students. Here students evaluate course workload and level, among other qualities, based on a course-agnostic set of questions.

One of these questions relates to the feedback that students get about their work; this is the question:

> In my opinion, I have received relevant academic feedback on my oral and written work on the course.[12]

[12] In Danish: "Jeg synes, at jeg har fået relevant faglig respons på mit skriftlige og mundtlige arbejde på kurset."

	2017	2018	2019	2020	2021	2022	2023
Strongly disagree	12.5%	5.8%	4.6%	0.0%	5.4%	2.6%	4.9%
Disagree.	10.7%	11.6%	4.6%	9.8%	7.1%	0.0%	12.2%
Neither agree nor disagree	17.9%	14.5%	12.3%	17.6%	17.9%	18.4%	7.3%
Agree	32.1%	37.7%	43.1%	25.5%	32.1%	42.1%	31.7%
Strongly agree	26.8%	30.4%	35.4%	47.1%	37.5%	34.2%	41.5%
Score	2.50	2.75	3.00	3.10	2.89	2.97	2.86
Number of answers	56	69	65	61	54	38	41

Table 2. Evaluation on CompSys of the question "In my opinion, I have received relevant academic feedback on my oral and written work on the course." The score is calculated between 0 and 4, 4 being "Strongly agree", as the sum-product of the score and percentage.

In CompSys, it is mainly the TAs that communicate directly with the students—thus, TAs are the primary feedback-givers. The teachers mainly do lectures, but also sometimes interact with students at exercise classes and in an online forum.

Table 2 shows the result of the evaluations along this question. We have not included data for 2016. This was the first year for the newly designed course and had many significant problems; this is therefore considered as an outlier. In 2017 the course was redesigned to the format that is still used today.

Unfortunately, we do not have data from before starting to use `remarks` and can therefore not make a comparative study. However, `remarks` was a continuous development project and increasingly integrated into the course from 2016 to 2019. We can see a general upwards trend over the integration period with less than 10% being dissatisfied. During and after the Covid-19 shutdown the data show a less clear picture; the number of students evaluations also falls.

5.3 TA Evaluations

To get an assessment of `remarks` we have asked former TAs on both CompSys and ITS to fill out a questionnaire. The questionnaire is was separated into three parts and all questions are included in Appendix B:

- Simple questions that follows the System Usability Scale (SUS) as defined by the ISO standard, ISO 9241 Part 11 [1, 6].
- Assessment based on prior TA experiences from other courses.
- Assessment based on subsequent TA experiences from other courses.

The questionnaire was distributed to about 25 to 30 former TAs going all the way back to 2019. Of these 11 filled the questionnaire. The call was only distributed on university e-mails addresses; some of the older TAs have finished their studies and might not check this address. It is therefore expected that answers mainly were from the more recent TAs.

System Usability Scale The first part of the questionnaire was a specialisation to the questions of SUS. The result of this part can be seen in Table 3.

SUS Question	Average
1. I like to use `remarks` to assist in marking student work.	3.55
2. I found `remarks` unnecessarily complex.	0.64
3. I thought `remarks` was easy to use.	3.45
4. I needed support from a teacher to be able to use `remarks`.	1.18
5. I found that the various functions in `remarks` were well integrated.	2.45
6. I thought there was too much inconsistency in `remarks`.	0.91
7. I would imagine that most people would learn to use `remarks` very quickly.	3.09
8. I found `remarks` very cumbersome to use.	0.64
9. I felt very confident using `remarks`.	3.18
10. I needed to learn a lot of things before I could get going with `remarks`.	0.55
SUS score average:	79.5%
SUS score minimum:	65.0%
SUS score maximum:	97.5%

Table 3. SUS questions with average and SUS score of questionnaire to TAs. Each question is assessed from 0 to 4, with 0 being "strongly disagree" and 4 being "strongly agree". The SUS score is a percentage.

The data shows that the TAs generally have a strong preference for answers that gives the best grading of `remarks`. The two the question with the lowest assessment are Questions 4 and 5. Question 4 is about how much assistance that the user needs to use `remarks` and Question 5 is about the integration of the tool. This was quite expected as `remarks` has been under development and the user interface has been less prioritised.

The average SUS score is 79.5% with a minimum of 65.0% and maximum of 97.5%. The SUS score is expected to follow the grading scale of a norm-referenced test, which means that average can be interpreted as a grade B with a spread from grades C to A. Of course, to conclude this for certain we would need to evaluate many other teaching evaluation tools. This is outside the scope of this paper, but we will conclude that the TAs see `remarks` quite favourably.

In Relation to Prior and Consecutive Experience Of the 11 TAs that answered the questionnaire, 4 TAs had experience with assessment prior to the courses. 3 of the TAs reported that they had gotten "none" to "simple descriptions" for marking on these courses. 1 TA reported that they had gotten "detailed descriptions". 2 TAs reported that `remarks` gave better support, while the other two TAs reported "not sure".

Of the 11 TAs that answered the questionnaire, 6 TAs got subsequent experience on other courses. Of these 4 of the TAs reported that they had gotten "none" to "simple descriptions" for marking on these courses; 1 TA reported getting only "oral descriptions". Only 1 TA reported that they had gotten "detailed descriptions". 4 TAs reported that `remarks` gave better support, while the other two TAs reported "not sure".

This is a small sample-size, but the results indicate that there is a lack of systematic support for TA work and remarks can be a tool for this. It could be interesting to compare this with the support TAs get across the department.

5.4 Personal Experience

To finish the experience section, the authors would like to give a personal account as course responsible for both CompSys and ITS over several years. The authors started the work on remarks to mitigate some of the work managing many TAs and ensuring better and consistent assessment of student work. From that perspective, we find that remarks has been quite successful.

remarks has given clear understanding between teachers and TAs on the courses, where we talk more about general assessment of the assignments and very little about individual hand-ins. Very few students have been complaining about the assessment they have received.

For the grading of the exams it has also been a very useful tool. This mainly shows in two ways. Firstly, the external examiners have expressed great satisfaction with the level of detail they get. The clear evaluative criteria mean that there are very few students that need to be discussed at a final grading meeting. This have lately taken less than an hour for grading about 160 students. Secondly, we have received very few exam complaints. Since the 2018 CompSys have gotten 2 formal complaints and 4 informal requests to consider the grade. The informal requests and one of the formal exam complaints were due to errors in viewing pdf-files. Having the remarks files made it easy to go back to the assessment to spot and correct the error.

The main downside of today is that remarks requires some technical skill to setup and manage. However, this also give a large amount of flexibility. This is something that will be addressed in future work.

6 Conclusion

In this paper we have introduced and given an assessment of remarks; an open-source suite of tools for marking student work. We have described how it can be used for marking semi-structured written student work in fair-sized Computer Science courses.

We have described how remarks has been used on two bachelor courses at the University of Copenhagen and detailed the scenarios in which TAs are aided to perform student assessment. The data that have been gathered from this experience, shows that remarks has a positive effect. We have indications that students are satisfied with the feedback that they get and that the TAs like to have the support that they get from remarks.

It is clear that remarks is still an academic software development project. To that end, future work can be classified into three areas. Firstly, we would like to get experience with using remarks in more courses. There are other courses at the University of Copenhagen that have started to use remarks, which is a

step forward. However, this leads to the second issue. To have wider adaptation, we need to make `remarks` easier to use. A web-based interface is a first step. Advanced uses of `remarks` today as a teacher still require some level of shell scripting, which speaks to shortcomings of the tool suite. Finally, we would like to more formally evaluate `remarks` wrt. other approaches.

We invite you to check out our project on GitHub; get in touch with the authors if you would like to try out `remarks`, or would like to argue that your spreadsheet, built over years of assessment experience, is superior to `remarks`.

References

1. 9241-210:2019, I.: Ergonomics of human-system interaction – part 210: Human-centred design for interactive systems (2019), https://www.iso.org/standard/77520.html
2. Brown, G.A., Bull, J., Pendlebury, M.: Assessing student learning in higher education. Routledge, 1 edn. (1997). https://doi.org/10.4324/9781315004914
3. Bull, J., McKenna, C.: Blueprint for computer-assisted assessment. Routledge, 1 edn. (2003). https://doi.org/10.4324/9780203464687
4. Conole, G., Warburton, B.: A review of computer-assisted assessment. Research in Learning Technology **13**(1), 17–31 (2005). https://doi.org/10.1080/0968776042000339772
5. Dawson, P.: Assessment rubrics: towards clearer and more replicable design, research and practice. Assessment & Evaluation in Higher Education **42**(3), 347–360 (2017). https://doi.org/10.1080/02602938.2015.1111294
6. Lewis, J.R.: The system usability scale: Past, present, and future. International Journal of Human–Computer Interaction **34**(7), 577–590 (2018). https://doi.org/10.1080/10447318.2018.1455307
7. Macedo, J.N., Moreira, R., Cunha, J., Saraiva, J.: Get your spreadsheets under (version) control. Ibero-American Conference on Software Engineering (2019)
8. Pickering, R.: Language-Oriented Programming, pp. 327–349. Apress, Berkeley, CA (2009). https://doi.org/10.1007/978-1-4302-2390-0_12
9. Popham, W.J.: What's wrong-and what's right-with rubrics. Educational leadership **55**, 72–75 (1997)
10. Sadler, D.R.: Indeterminacy in the use of preset criteria for assessment and grading. Assessment & Evaluation in Higher Education **34**(2), 159–179 (2009). https://doi.org/10.1080/02602930801956059
11. Secolsky, C., Dension, D.B. (eds.): Handbook on Measurement, Assessment, and Evaluation in Higher Education. Routledge, 2 edn. (2018). https://doi.org/10.4324/9781315709307
12. Terry J. Crooks, M.T.K., Cohen, A.S.: Threats to the valid use of assessments. Assessment in Education: Principles, Policy & Practice **3**(3), 265–286 (1996). https://doi.org/10.1080/0969594960030302
13. Ward, M.P.: Language-oriented programming. Software - Concepts and Tools **15**(4), 147–161 (1994)
14. Yanakieva, E., Bird, P., Bieniusa, A.: A study of semantics for crdt-based collaborative spreadsheets. In: Proceedings of the 10th Workshop on Principles and Practice of Consistency for Distributed Data. p. 37–43. PaPoC '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3578358.3591324

A Remarks example from Computer Systems

Example of a detailed `remarks` file for the first assignment on the Computer Systems course at DIKU. It is written by the teachers for TAs that correct the assignments. In the assignment the students are to implement a simple version of `file(1)` in C.

```
# Feedback:

Corrected by: [Name]

  [Your Feedback]


# API: /10
  * Can read path from command-line.
  * Writes to stdout even if file does not exist.
  * Writes to stderr if no arguments are provided.
  * Exit code is 0, even if file does not exist.
  * Exit code is 1, if no arguments are provided.


# File Types: /15

## Implementation: /10
  * Understand EXIT_SUCCESS, EXIT_FAILURE.
  * Can open a file.
  * Can handle non-existing paths.
  * Understand errno.
  * Can read a file one byte a time.
  * Understand bit-wise operators.
  * Understand bit masking.
  * Understand ASCII.
  * Can write a for-loop and/or while-loop.
  * Can write a switch-case and/or if statement.
  * Can read multiple bytes in sequence (UTF-8).
  * Understands variable-width encoding.
  * Remember to close the file.
  * Do fallback to data if file contains spurious null-bytes.


## Bonus: +0
  * Support null-bytes at the end of a file.

## Style: /5
  * Check return codes throughout.
```

```
  * Don't have magic constants.
  * Functions are short, concise, and well-named.
  * The style is sensible in general.
  * No zip bomb
  * src/ is located correctly


# Testing: /25
  * Understand echo.
  * Understand hex.
  * Understand what a shell script is.
  * Understand diff.
  * There are files for testing all different types.
  * There are files for testing empty.
  * There are files for testing data.
  * There are (automated) tests.
  * Tests are reproducible.
  * Test-results are comprehensible.
  * Tests are extensible.


# Report: /40
  * Report is comprehensible.
  * Gives a good overview of what has been solved.
  * Actually describes the submitted code.
  * Describes the added tests.
  * Discusses the non-trivial parts.
  * Disambiguates properly (e.g., trailing null-byte).
  * Contains both name and KUid
  * Describes what file(1) reports for a binary.


# Theory: /10

## Boolean logic: /2
  * Understand Boolean logic.
  * It is the same, as both sides in some way say "set the bits
    to 1 for which A has 1 and B has 0".

## Logical operators: /6
  * Understands bit-wise operators
  * Understands bit masking
  * Can show the expression
    * x << 3 -- uses shift, note 3 and not 8
    * !(x ^ 0x6) -- uses xor for comparison
```

```
    * (x >> 31) || !x -- property of two's complement numbers or 0
    * !(!(x ^ y)) eller (x^y) (venstre giver 1 eller 0, højre giver
      non-0 eller 0) -- similiar to exercise 2

## Floating point: /2
  * Understand floating points
  * Can explain what denormalised numbers are (used when exp == 0),
    makes M have a leading 0 (before the fraction point, in normal
    numbers it has a leading 1)
  * Knows advantage (can encode 0 that is 0x0, can represent values
    closer to 0)
```

B Questionnaire questions

Questions based on the System Usability Scale assessed from "Strongly Disagree" to "Strongly Agree":

- I like to use Remarks to assist in marking student work.
- I found Remarks unnecessarily complex.
- I thought Remarks was easy to use.
- I needed support from a teacher to be able to use Remarks.
- I found that the various functions in Remarks were well integrated.
- I thought there was too much inconsistency in Remarks.
- I would imagine that most people would learn to use Remarks very quickly.
- I found Remarks very cumbersome to use.
- I felt very confident using Remarks.
- I needed to learn a lot of things before I could get going with Remarks.

Questions relation to prior experience:

- Did you mark student work on courses before you were introduced to Remarks? (yes/no)
- What has generally been your role in these previous courses?
  - Teaching Assistant
  - Head Teaching Assistant
  - Teacher
  - External Examiner
  - Other
- What marking support did you get on previous course:
  - None
  - Simple Description
  - Oral Description
  - Detailed Description
  - Remarks
- Did you find that Remarks gave you better support in the marking process? (yes/no/not sure)

- What did you find positive about this prior experience? (Text field)
- What did you find negative about this prior experience? (Text field)

Questions relation to subsequent experience:

- Did you mark student work on courses before you were introduced to Remarks? (yes/no)
- What has generally been your role in these subsequent courses?
  - Teaching Assistant
  - Head Teaching Assistant
  - Teacher
  - External Examiner
  - Other
- What marking support did you get on subsequent course:
  - None
  - Simple Description
  - Oral Description
  - Detailed Description
  - Remarks
- Did you find that Remarks gave you better support in the marking process? (yes/no/not sure)
- What did you find positive about this subsequent experience? (Text field)
- What did you find negative about this subsequent experience? (Text field)

C Language-Oriented Programming

Language-oriented programming is an approach to software development where we principally design a domain-specific language for representing and solving problems in the target domain of the software [8, 13]. The language may, but does not need to be exposed to the end-user. It is intended mainly as a domain of discourse for the developers of the software.

One benefit of this approach is that we can arrive at a maximally concise and safe domain modeling language. In contrast, if we begin by modeling in a general-purpose programming language, we are forced to use, and are constrained by the constructs and connectives afforded by that language. This may limit our ability to safely and succinctly model the domain from both a syntactic and semantic perspective. Herein lies the second benefit, as the practice of formally modeling programming language syntax and semantics is much more well-established than the practice of formally modelling software, in general.

The downside of this approach is indeed that we develop a near full-blown programming language. This can seem like a lot of work for the uninitiated. However, today there exists a wealth of domain-specific language workbenches, and a number of general-purpose programming languages have been expressly designed to streamline creation of embedded and extraneous domain-specific programming languages (e.g., Haskell, Racket).

In section 3 we use BNF-style notation to introduce the domain model of `remarks`. The domain model closely corresponds the abstract syntax tree in our implementation. It is intentional that there is a close correspondence between the two, so Section 3 reveals only an *abstract* syntax for `remarks`. For instance, we do not specify how many spaces are allowed between the tokens.

Section 4 reveals the details of the concrete syntax of our current plain-text format. However, the abstract syntax introduced in Section 3 may also well be concretely implemented as a data model in a database management system. In that case, details about whitespace, for instance, would have be obtrusive.

Since there are many BNF-style notations, we clarify our conventions here: An italicized, CamelCased name in angle brackets (e.g., $\langle \mathit{Judgement} \rangle$) denotes a syntactic non-terminal. A text in monospaced font enclosed in single quotation marks (e.g., '`Analytic`') denotes a syntactic terminal (i.e., literal or atomic value). Syntactic non-terminals may be indexed by non-negative integers (e.g., $\langle \mathit{Judgement} \rangle_1$). A non-terminal is declared either in prose or by using the declaration operator '`::=`', with a pattern on the left-hand side, and a non-empty list of projections on the right-hand side, separated by '`|`':

$$\langle \mathit{Pattern} \rangle \text{ `::=' } \langle \mathit{Projection} \rangle_1 \text{ `|' } \langle \mathit{Projection} \rangle_2 \text{ `|' } \cdots \text{ `|' } \langle \mathit{Projection} \rangle_n$$

A $\langle \mathit{Pattern} \rangle$ names an undeclared non-terminal, and may be indexed by a variable name, meaning that we are defining an indexed non-terminal. The variable name becomes a fresh name which can be used in the projections on the right-hand side of the declaration.

Finally, a non-terminal occurring on the right-hand side of a declaration, may be qualified with Kleene-star denoting 0 or more occurrences of the non-terminal (e.g., $\langle \mathit{Judgement} \rangle_1^*$).

Section 4 provides the concrete plain-text syntax of `remarks`, as understood by our tool-suite today. This section also uses BNF-style notation, but to distinguish this concrete syntax from the abstract syntax in Section 3, Section 4 uses double angle brackets for the non-terminals (e.g., $\langle\!\langle \langle \mathit{Judgement} \rangle_n \rangle\!\rangle$).

D File System and Command-Line Interface

`remarks` was principally designed to streamline asynchronous collaboration on assessments among dozens assessors, make conflict resolution explicit and unsurprising, and minimize possibility of conflicting edits, in general.

To that end, `remarks` files are plain-text files (as discussed in Section 4), amenable to revision control with a conventional source code revision control system (e.g., Git). Furthermore, we have usually used one `remarks` file per unit of work to be assessed. Notably, a "unit of work" does not necessarily correspond to "a submission". For instance, it may be the theory or practice part of a submission, each assessed by a different assessor. By virtue of planning, it is almost never the case that two assessors are assessing the same unit of

work at the same time. Hence, assessors rarely, if ever face merge conflicts when synchronizing their assessments.

The `remarks` tool suite consists of a single command-line utility called `remarks`. This tool can be used to arrive at a summative or descriptive assessments based on a `remarks` file. For instance:

`check` checks that a `remarks` file has been filled in correctly.
`summary` checks and prints a summary of the points, down to a given depth.
`pending` prints what parts are yet to be assessed.
`export` exports judgements in given format (e.g., HTML, Markdown, PdfMark).

E Example: Org-mode

Another popular choice of administrative software for conducting assessment is Emacs Org-mode. Using this tool requires some, though not intricate knowledge of the Emacs text-editor.

A typical assessment Org-mode file, reminiscent of our Table 1, may look something like this:

```
* S1
** Theory
*** T1
  :PROPERTIES:
  :points:
  :END:
  (free-form textual remarks)
*** T2
  :PROPERTIES:
  :points:
  :END:
  (free-form textual remarks)
...
```

Emacs can then be used to compute summative assessments analytically, based on the points given in the leaf `:points:` attributes. These can then be stored in the `.org` file itself.

This is considerably more verbose than `remarks`, making it hard to manage in a conventional source code revision control system. This file is easier to look at in Emacs, since it conveniently folds the properties and bullet-points. However, that functionality is of course not there when looking at a conventional Git diff.

It is quite possible that most `remarks` functionality can be achieved with Emacs Org-mode. However Emacs Org-mode will always be a general purpose admistrative software data format, while `remarks` strives to be an eloquent domain-specific language for marking student work.